Wireless Sensor Networks
Chapter 8: Time Synchronization

António Grilo

Courtesy: Holger Karl, UPB
Goals of this chapter

- Understand the importance of time synchronization in WSNs
- Understand typical strategies for time synchronization and how they are applied in WSNs
Overview

- The time synchronization problem
- Protocols based on sender/receiver synchronization
- Protocols based on receiver/receiver synchronization
- Summary
Overview

- The time synchronization problem
- Protocols based on sender/receiver synchronization
- Protocols based on receiver/receiver synchronization
- Summary
Example

- Goal: estimate angle of arrival of a very distant sound event using an array of acoustic sensors
- From the figure, $\theta$ can be estimated when $x$ and $d$ are known:

\[ x = d \sin \theta \]

- $d$ is known a priori, $x$ must be estimated from differences in time of arrival
  - $x = C \Delta t$ where $C$ is the speed of sound
  - For $d=1$ m and $\Delta t=0.001$ we get $\theta = 0.336$ radians
  - When $\Delta t$ is estimated with 500 $\mu$s error, the $\theta$ estimates can vary between 0.166 and 0.518
- Morale: a seemingly small error in time synch can lead to significantly different angle estimates
The role of time in WSNs

- Time synchronization algorithms can be used to better synchronize clocks of sensor nodes

- Time synchronization is needed for WSN applications and protocols:
  - Applications: AOA estimation, beamforming
  - Protocols: TDMA, protocols with coordinated wakeup, ...
  - Distributed debugging: timestamping of distributed events is needed to figure out their correct order of appearance

- WSN have a direct coupling to the physical world, hence their notion of time should be related to **physical time**:
  - physical time = wall clock time, real-time, i.e. one second of a WSN clock should be close to one second of real time
  - Commonly agreed time scale for real time is UTC, generated from atomic clocks and modified by insertion of leap seconds to keep in synch with astronomical timescales (one rotation of earth)
  - Other concept: logical time (Lamport), where only the relative ordering of events counts but not their relation to real time
Clocks in WSN nodes

- Often, a **hardware clock** is present:
  - An oscillator generates pulses at a fixed nominal frequency
  - A counter register is incremented after a fixed number of pulses
    - Only register content is available to software
    - Register change rate gives achievable time resolution
  - Node i’s register value at real time t is $H_i(t)$

  - Convention: small letters (like t, t’) denote real physical times, capital letters denote timestamps or anything else visible to nodes

- A (node-local) software clock is usually derived as follows:
  - $L_i(t) = \theta_i H_i(t) + \phi_i$ (not considering overruns of the counter-register)
  - $\theta_i$ is the (drift) rate, $\phi_i$ the phase shift
  - Time synchronization algorithms modify $\theta_i$ and $\phi_i$, but not the counter register
Synchronization accuracy / agreement

- **External synchronization:**
  - synchronization with external real time scale like UTC
  - Nodes $i=1, \ldots, n$ are accurate at time $t$ within bound $\delta$ when $|L_i(t) - t| < \delta$ for all $i$
    - Hence, at least one node must have access to the external time scale

- **Internal synchronization**
  - No external timescale, nodes must agree on common time
  - Nodes $i=1, \ldots, n$ agree on time within bound $\delta$ when $|L_i(t) - L_j(t)| < \delta$ for all $i,j$
Sources of inaccuracies

- Nodes are switched on at random times, phases $\theta_i$ hence can be random
- Actual oscillators have random deviations from nominal frequency (drift, skew)
  - Deviations are specified in ppm (pulses per million), the ppm value counts the additional pulses or lost pulses over the time of one million pulses at nominal rate
  - The cheaper the oscillators, the larger the average deviation
    - For sensor nodes values between 1 ppm (one second every 11 days) and 100 ppm (one second every 2.8 hours) are assumed, Berkeley motes have an average drift of 40 ppm
- Oscillator frequency depends on time (oscillator aging) and environment (temperature, pressure, supply voltage, ...)
  - Especially the time-dependent drift rates call for frequent re-synchronization, as one-time synchronization is not sufficient
  - However, stability over tens of minutes is often a reasonable assumption
General properties of time synchronization algorithms

- Physical time vs. logical time
- External vs. internal synchronization
- Global vs. local algorithms
  - Keep all nodes of a WSN synchronized or only a local neighbourhood?
- Absolute vs. relative time
- Hardware vs. software-based mechanisms
  - A GPS receiver would be a hardware solution, but often too heavyweight/costly/energy-consuming in WSN nodes, and in addition a line-of-sight to at least four satellites is required
- A-priori vs. a-posteriori synchronization
  - Is time synchronization achieved before or after an interesting event? ➔ Post-facto synchronization
- Deterministic vs. stochastic precision bounds
- Local clock update discipline
  - Should backward jumps of local clocks be avoided? (Users of make say yes here ....)
  - Avoid sudden jumps?
Performance metrics and fundamental structure

- Metrics:
  - Precision: maximum synchronization error for deterministic algorithms, error mean / stddev / quantiles for stochastic ones
  - Energy costs, e.g. # of exchanged packets, computational costs
  - Memory requirements
  - Fault tolerance: what happens when nodes die?

- Fundamental building blocks of time synchronization algorithms:
  - Resynchronization event detection block: when to trigger a time synchronization round? Periodically? After external event?
  - Remote clock estimation block: figuring out the other nodes clocks with the help of exchanging packets
  - Clock correction block: compute adjustments for own local clock based on estimated clocks of other nodes
  - Synchronization mesh setup block: figure out which node synchronizes with which other nodes
Constraints for Time Synchronization in WSNs

- An algorithm should scale to large networks of unreliable nodes
- Quite diverse precision requirements, from ms to tens of seconds
- Use of extra hardware (like GPS receivers) is mostly not an option
- Low mobility
- Often there are no fixed upper bounds on packet delivery times (due to MAC delays, buffering, ...)
- Negligible propagation delay between neighboring nodes
- Manual node configuration is not an option
Post-facto synchronization

- Basic idea:
  - Keeping nodes synchronized all the time incurs substantial energy costs due to need for frequent resynchronization
    - Especially true for networks which become active only rarely
  - When a node observes an external event at time $t$, it stores its local timestamp $L_i(t)$, achieves synchronization with neighbor node / sink node and converts $L_i(t)$ accordingly
- Can be implemented in different ways, to be discussed later
Overview

- The time synchronization problem
- Protocols based on sender/receiver synchronization
- Protocols based on receiver/receiver synchronization
- Summary
Protocols based on sender/receiver synchronization

- In this kind of protocols, a receiver synchronizes to the clock of a sender
- We have to consider two steps:
  - Pairwise synchronization: how does a single receiver synchronize to a single sender?
  - Networkwide synchronization: how to figure out who synchronizes with whom to keep the whole network / parts of it synchronized?
- The classical NTP protocol [Mills, RFC 1305] belongs to this class
LTS – Lightweight Time Synchronization

- Overall goal: synchronize the clocks of all sensor nodes / of a subset of nodes to one reference clock (e.g. equipped with GPS receivers)
- It allows to synchronize the whole network, parts of it and it also supports post-facto synchronization
- It considers only phase shifts and does not try to correct different drift rates
- Two components:
  - pairwise synchronization: based on sender/receiver technique
  - networkwide synchronization: minimum spanning tree construction with reference node as root
LTS – Pairwise Synchronization

- Trigger resynchronization
- Format synch packet
- Timestamp packet with $L_i(t_1)$
- Hand over packet for transmission
- Operating system, channel access
- Start packet transmission
- Propagation delay
- Packet transmission time
- Packet reception interrupt
- Timestamp with $L_j(t_5)$
- Format synch answer packet
- Timestamp with $L_j(t_6)$
- Hand over packet for transmission
- OS, Channel access
- Start packet transmission
- Packet reception interrupt
- Timestamp with $L_i(t_8)$
LTS – Pairwise Synchronization

- Node i wants to synchronize its clock to node j’s clock
- Timeline:
  - Node i triggers resynchronization at time $t_0$, formats packet, timestamps it at $t_1$ with $L_i(t_1)$ and hands it over to transmission (with $L_i(t_1)$ as payload)
  - At $t_2$ the first bit appears on the channel, at $t_3$ the receiver receives last bit, packet reception is signaled at $t_4$, and at $t_5$ node j timestamps it with $L_j(t_5)$
  - Node j formats answer packet, timestamps it at time $t_6$ with $L_j(t_6)$ and hands it over for transmission – as payload the timestamps $L_i(t_1)$, $L_j(t_5)$ and $L_j(t_6)$ are included
  - The arrival of the answer packet is signaled at time $t_7$ to node i, and i timestamps it afterwards with $L_i(t_8)$
- After time $t_8$, node i possesses four values: $L_i(t_1)$, $L_j(t_5)$, $L_j(t_6)$ and $L_i(t_8)$ and wants to estimate its clock offset to node j – but how?
LTS – Pairwise Synchronization

- Assumptions:
  - Node i’s original aim is to estimate the true offset \( O = \Delta(t_1) = L_i(t_1) - L_j(t_1) \)
  - During the whole process the drift is negligible
    - the algorithm in fact estimates \( \Delta(t_5) \) and assumes \( \Delta(t_5) = \Delta(t_1) \)
  - Propagation delay \( \tau \) the same in both directions, request and answer packets have duration \( t_p \), both parameters are known to i

- Approach:
  - Node i estimates \( \Delta(t_5) = L_i(t_5) - L_j(t_5) \) and therefore needs to estimate \( L_j(t_5) \), which is generated “somewhere” between \( t_1 \) and \( t_8 \)
  - When \( t_8 - t_1 \) is very small, we might be willing to approximate \( O \approx L_i(t_1) - L_j(t_5) \) or as \( \frac{1}{4} L_i(t_8) - L_j(t_5) \)
  - But we can reduce uncertainty:
    - after \( t_1 \) we have at least one propagation delay and packet transmission time (for the request packet)
    - before \( t_8 \) we have another propagation delay and packet transmission time (for the response packet)
    - There passes also time between \( L_j(t_5) \) and \( L_j(t_6) \), and \( L_i(t_5) \) must be before this interval
LTS – Pairwise Synchronization

- Under the assumption that the remaining uncertainty is allocated equally to both i and j, node i can estimate $L_j(t_5)$ as

$$L_i(t_5) = \frac{L_i(t_1) + \tau + t_p + L_i(t_8) - \tau - t_p - (L_j(t_6) - L_j(t_5))}{2}$$

- This means:

$$0 = \Delta(t_5) = L_i(t_5) - L_j(t_5) = \frac{L_i(t_8) + L_i(t_1) - L_j(t_6) - L_j(t_5)}{2}$$
LTS – Pairwise Synchronization -- Discussion

- Node i can figure out the offset to node j based on the known values $L_i(t_1)$, $L_j(t_5)$, $L_j(t_6)$, $L_i(t_8)$

- This exchange takes two packets – if node j should also learn about the offset, a third packet is needed from i to j carrying O

- The uncertainty is in the interval
  
  $I = [L_i(t_1)+\tau+t_p, L_i(t_8) - \tau - t_p - (L_j(t_6) - L_j(t_5))]$

  and by picking the mid-point of the interval as $L_i(t_5)$, the maximum uncertainty is $|I|/2$
LTS – Pairwise Synchronization -- Discussion

• Sources of inaccuracies:
  • MAC delay
  • interrupt latencies upon receiving packets
  • Delays between packet interrupts and timestamping operation
  • Delay in operating system and protocol stack

• Improvements:
  • Let i timestamp its packet after the MAC delay, immediately before transmitting the first bit
    • This would remove the uncertainty due to i’s operating system / protocol stack and the MAC delay!!
  • Let j timestamp receive packets as early as possible, e.g. in the interrupt routine
    • this removes the delay between packet interrupts and timestamping from the uncertainty, and leaves only interrupt latencies

→ these things are hard to do when COTS hardware is used, but easy when full source code of MAC and direct access to hardware are available
LTS – Pairwise Synchronization – Error Analysis

- Elson et al measured pairwise differences in timestamping times at a set of receivers when timestamping happens in the interrupt routine (Berkeley motes)
- Estimated distribution:
  - normal random variable (rv) with zero mean/stddev of 11.1 μs
- Additional assumption: uncertainty on the transmitter side has same distrib. and is independent
- Hence:
  - total uncertainty is a zero-mean normal rv with variance $4\sigma^2$
  - For a normal rv 99% of all outcomes have maximum distance of $2.3\sigma$ to mean
  - the maximum synchronization error is with 99% probability smaller than $2.3 \times 2 \times \sigma$
LTS – Networkwide Synchronization

- We are given one reference node $R$, to which all other nodes / a subset of nodes want to synchronize
  - $R$’s direct neighbors (level-1 neighbors) synchronize with $R$
  - Two-hop (level-2) neighbors synchronize with level-1 neighbors
  - ....
- This way a spanning tree is created
- But one should not allow arbitrary spanning trees:
  - Consider a node $i$ with hop distance $h_i$ to the root node
  - Assume that:
    - all synchronization errors are independent
    - all synch errors are identically normally distributed with zero mean and variance $4\sigma^2$
    - Then node $i$’s synchronization error is a zero-mean normal rv with variance $h_i 4\sigma^2$
    - Hence, a minimum spanning tree minimizes synchronization errors
LTS – Centralized Multihop LTS

- Reference node R triggers construction of a spanning tree, it first synchronizes its neighbors, then the first-level neighbors synchronize second-level neighbors and so on.

- Different distributed algorithms for construction of spanning tree can be used, e.g. DDFS, Echo algorithm.

- Communication costs:
  - Costs for construction of spanning tree
  - Synchronizing two nodes costs 3 packets, synchronizing n nodes costs 3n packets.
LTS – Distributed Multihop LTS

- No explicit construction of spanning tree needed, but each node knows identity of reference node(s) and routes to them.
- When node 1 wants to synchronize with R, an appropriate request travels to R – following this, 4 synchronizes to R, 3 synchronizes to 4, 2 synchronizes to 3, 1 synchronizes to 2.
  - By-product: nodes 2, 3, and 4 are synchronized with R.
- Minimum spanning tree constructed implicitly: node 1 should know shortest route to the closest reference node.
LTS – Distributed Multihop LTS -- Variations

- When node 5 wants to synchronize with R, it can:
  - issue its own synchronization request using route over 3, 4 and put additional synchronization burden on them
  - ask in its local neighborhood whether someone is synchronized or has an ongoing synchronization request and benefit from that later on
  - Enforce usage of path over 7, 8 (path diversification) to also synchronize these nodes

- Discussion:
  - Simulation shows that distributed multihop LTS needs more packets (between 40% and 100%) when all nodes have to be synchronized, even with optimizations
  - Distributed multihop LTS allows to synchronize only the minimally required set of nodes → post-facto synchronization
Other Sender-/Receiver-based Protocols

• These protocols work similar to LTS, with some differences in:
  • Method of spanning tree construction
  • How and when to take timestamps
  • How to achieve post-facto synchronization

• One variant: TPSN (Timing-Sync Protocol for Sensor Nets)
  • Pairwise-protocol similar to LTS, but timestamping at node i happens immediately before first bit appears on the medium, and timestamping at node j happens in interrupt routine
  • Spanning tree construction based on level-discovery protocol:
    • root issues level_discovery packet with level 0
    • neighbors assign themselves level 1 + level value from level_discovery
    • neighbors wait for some random time before they issue level_discovery packets indicating their own level
    • Nodes missing level_discovery packets for long time ask their neighborhood
TSync

- TSync combines:
  - HRTS (Hierarchy Referencing Time Synchronization): a protocol to synchronize a broadcast domain to one of its members
  - ITR (Individual-based Time Request): a sender-/receiver protocol similar to LTS/TPSN
  - A networkwide synchronization protocol
- HRTS provides a technique to synchronize a group of nodes to a reference node with only three packets!
HRTS

Timestamp with $L_i(t_2)$

Timestamp answer with $L_i(t_3)$

$\text{sync\_begin}(\text{level}, i)$

Compute offset $O$

$(O, L_i(t_2))$

$(O, L_i(t_2))$
HRTS

- Node i and j want to synchronize to R’s clock, but cannot hear each other
- Timeline:
  - Root node triggers time synchronization at time $t_1$ at local time $L_R(t_1)$, it formats a sync_begin packet and includes i’s address and a level value
  - Node i timestamps packet at time $t_2$ with $L_i(t_2)$ and node j timestamps it at $t_2'$ with $L_j(t_2')$
  - Node i formats an answer packet and timestamps it at time $t_3$ with $L_i(t_3)$ – the packet includes the values $L_i(t_2)$ and $L_i(t_3)$
  - Root node R timestamps the answer packet at time $t_4$ with $L_R(t_4)$ and computes its offset $O_{R,i}$ with node i’s clock (it can do this similar to LTS)
  - Root node R formats another packet including the values $O_{R,i}$ and $L_i(t_2)$ and broadcasts this packet
- Node j possesses the values $L_j(t_2')$, $O_{R,i}$ and $L_i(t_2)$ and wants to determine $O_{R,j}$ out of these values – but how?
  - Node i already knows its offset $O_{i,R}$ when it has received $O_{R,i}$
Node j’s evaluation is based on the assumption that $t_2 = t_2'$
  - reasonable when both nodes timestamp in their interrupt routines and propagation delay is small

Please note that under this assumption we have:

- $L_i(t_2) = L_j(t_2') + O_{j,i} \implies O_{j,i} = L_i(t_2) - L_j(t_2')$

- $L_i(t_2) = L_R(t_2) + O_{R,i}$

- $L_j(t_2') = L_R(t_2) + O_{R,j}$

This gives:

- $O_{R,j} = L_j(t_2') - L_R(t_2) = L_j(t_2') - (L_i(t_2) - O_{R,i})$
HRTS -- Discussion

- Node $j$ is not involved in any packet exchange ➔ by this scheme it is possible to synchronize an arbitrary number of nodes to R’s clock with only three packets!!

- The synchronization uncertainty comes from:
  - The error introduced by R when estimating $O_{R,i}$
  - The error introduced by setting $t_2 = t_2'$
    - This makes HRTS only feasible for geographically small broadcast domains

- Both kinds of uncertainty can again be reduced by:
  - timestamping outgoing packets as lately as possible (relevant for $t_1$ and $t_3$)
  - timestamping incoming packets as early as possible (relevant for $t_2$, $t_2'$, $t_4$

- The authors propose to use extra channels for synchronization traffic (control_channel for sync_begin, clock_channel for answer packets) when late timestamping of outgoing packets is not an option
  - Rationale: keep MAC delay small
It is assumed that some reference nodes are present in the network, e.g. having a GPS receiver

Initialization:
- Reference nodes assign themselves a level of 0
- All other nodes assign themselves a level of 1
- The reference node becomes a root node and synchronizes its neighbors

Whenever any node receives a sync_begin packet with a smaller level \( x \) than its current level \( y \):
- It synchronizes to the issuing node
- It assigns itself a level \( y := x + 1 \)
- It synchronizes its neighbors

This way a minimal spanning tree is constructed
Overview

- The time synchronization problem
- Protocols based on sender/receiver synchronization
- Protocols based on receiver/receiver synchronization
- Summary
Protocols based on receiver/receiver synchronization

- In this class of schemes the receivers of packets synchronize among each other, not with the transmitter of the packet
- RBS = Reference Broadcast Synchronization (Elson et al)
- RBS has two components:
  - Synchronize receivers within a single broadcast domain
  - A scheme for relating timestamps between nodes in different domains
- RBS does not modify the local clocks of nodes, but computes a table of conversion parameters for each peer in a broadcast domain
- RBS allows for post-facto synchronization
RBS – Synchronization in a Broadcast Domain

Packet reception interrupt

Packet reception interrupt

Receiver uncertainty

Timestamp with $L_i(t_{3,i})$

Timestamp with $L_j(t_{3,j})$

Send ($L_i(t_{3,i}), R, s$)

Send ($L_j(t_{3,j}), R, s$)
RBS – Synchronization in a Broadcast Domain

- The goal is to synchronize i’s and j’s clocks to each other
- Timeline:
  - Reference node R broadcasts at time $t_0$ some synchronization packet carrying its identification R and a sequence number $s$
  - Receiver i receives the last bit at time $t_{1,i}$, gets the packet interrupt at time $t_{2,i}$ and timestamps it at time $t_{3,i}$
  - Receiver j is doing the same
  - At some later time node i transmits its observation $(L_i(t_{3,i}), R, s)$ to node j
  - At some later time node j transmits its observation $(L_j(t_{3,j}), R, s)$ to node i
  - The whole procedure is repeated periodically, the reference node transmits its synchronization packets with increasing sequence numbers
    - R could also use ordinary data packets as long as they have sequence numbers ...
  - Under the assumption $t_{3,i} = t_{3,j}$ node j can figure out the offset $O_{i,j} = L_j(t_{3,j}) - L_i(t_{3,i})$ after receiving node i’s final packet – of course, node i can do the same
RBS – Synchronization in a Broadcast Domain

- The synchronization error in this scheme can have two causes:
  - There is a difference between $t_{3,i}$ and $t_{3,j}$
  - Drift between $t_{3,i}$ and the time where node $i$ transmits its observations to $j$

- But:
  - In small broadcast domains and when received packets are timestamped as early as possible the difference between $t_{3,i}$ and $t_{3,j}$ is very small
    - As compared to sender-/receiver based schemes the MAC delay and operating system delays experienced by the reference node play no role!!
  - Drift can be neglected when observations are exchanged quickly after reference packets
  - Drift can be estimated jointly with Offset $O$ when a number of periodic observations of $O_{i,j}$ have been collected
    - This amounts to a standard least-squares line regression problem
RBS – Synchronization in a Broadcast Domain

- Remember?
- Elson et al measured pairwise differences in timestamping times at a set of receivers when timestamping happens in the interrupt routine (Berkeley motes)
- This is just the distribution of the differences $t_{3,i} - t_{3,j}$!
Wireless Sensor Networks: Time Synchronization

RTS – Synchronization in a Broadcast Domain

- Communication costs:
  - Be \( m \) the number of nodes in the broadcast domain
  - First scheme: reference node collects the observations of the nodes, computes the offsets and sends them back \( \Rightarrow 2m \) packets
  - Second scheme: reference node collects the observations of the nodes, computes the offsets and keeps them, but has responsibility for timestamp conversions and forwarder selection \( \Rightarrow m \) packets
  - Third scheme: each node transmits its observation individually to the other members of the broadcast domain \( \Rightarrow m(m-1) \) packets
  - Fourth scheme: each node broadcasts its observation \( \Rightarrow m \) packets, but unreliable delivery

- Collisions are a problem:
  - The reference packets trigger all nodes simultaneously to tell the world about their observations

- Computational costs: least-squares approximation is not cheap!
RBS – Network Synchronization
RBS – Network Synchronization

- Suppose that:
  - node 1 has detected an event at time $L_1(t)$
  - the sink is connected to a GPS receiver and has UTC timescale
  - node 1 wants to inform the sink about the event such that the sink receives a timestamp in UTC timescale
  - Broadcast domains are indicated by “circles”

- Timestamp conversion approach:
  - Idea: do not synchronize all nodes to UTC time, but convert timestamps as packet is forwarded from node 1 to the sink $\Rightarrow$ avoids global synch!!!
  - Node 1 picks node 3 as forwarder – as they are both in the same broadcast domain, node 1 can convert the timestamp $L_1(t)$ into $L_3(t)$
  - Node 3 picks node 5 in the same way
  - Node 5 is member in two broadcast domains and knows also the conversion parameters for the next forwarder 9
  - And so on ...
  - Result: the sink receives a timestamp in UTC timescale!
  - Nodes 5, 8 and 9 are gateway nodes!
RBS – Network Synchronization

- **Forwarding options:**
  - Let each node pick its forwarder directly and perform conversion, the reference nodes act as mere pulse senders.
  - Let each node transmit its packet with timestamp to reference node, which converts timestamp and picks forwarder.
    - This way a broadcast domain is not required to be fully connected.
  - In either case the clock of the reference nodes is unimportant.

- **How to create broadcast domains?**
  - In large domains (large m) more packets have to be exchanged.
  - In large domains fewer domain-changes have to be made end-to-end, which in turn reduces synchronization error.
  - This is essentially a clustering problem, forwarding paths and gateways have to be identified by routing mechanisms.
Overview

• The time synchronization problem
• Protocols based on sender/receiver synchronization
• Protocols based on receiver/receiver synchronization
• Summary
Summary

- Time synchronization is important for both WSN applications and protocols.
- Using hardware like GPS receivers is typically not an option, so extra protocols are needed.
- Post-facto synchronization allows for time-synchronization on demand, otherwise clock drifts would require frequent re-synchronization and thus a constant energy drain.
- Some of the presented protocols take significant advantage of WSN peculiarities like:
  - small propagation delays
  - the ability to influence the node firmware to timestamp outgoing packets late, incoming packets early.
- Of course, there are many, many more schemes ....